

ASSEGNAMENTO DI RECUPERO: ALBERI TERNARI DI RICERCA

Informatica, Corso di Laurea in Fisica, Università di Pisa

AA 2017/18

Dato un multi-insieme K di numeri interi (insieme con ripetizioni), vogliamo rappresentare K in un albero ternario di ricerca, la cui rappresentazione è definita come segue.

Il numero di nodi dell'albero è uguale alla cardinalità di K . In particolare, ogni nodo dell'albero ternario di ricerca contiene una chiave $x \in K^1$ e punta a tre sottoalberi L (left), M (mid) e R (right) con la seguente proprietà.

- Il sottoalbero radicato in L contiene tutte le chiavi in K maggiori di x .
- Il sottoalbero radicato in M contiene tutte le chiavi in K uguali a x .
- Il sottoalbero radicato in R contiene tutte le chiavi in K minori di x .

Si noti che per costruzione, tutti i nodi nel sottoalbero M sono tali che L e R sono vuoti (per cui M è concettualmente una lista).

La definizione del tipo nodo viene data nel file *ttree.h* come segue.

```
typedef struct nodo {  
    /** chiave del nodo */  
    int key;  
    /* puntatori ai nodi figli */  
    struct nodo *left, *mid, *right;  
} nodo_t;
```

Tutti i nodi dell'albero hanno la stessa definizione. Per quanto detto sopra, tutti i nodi contenuti nel sottoalbero puntato da *mid* sono tali che *left* e *right* sono NULL. Dunque se un nodo x ha chiave 5 e suo padre y ha chiave 5, x deve avere *left* e *right* uguali a NULL per costruzione, come mostrato nell'esempio seguente.

Esempio Sia il multi-insieme K definito come segue: $\{12, 45, 56, 78, 4, 56, 9, 6, 9, -9, 3, -22, 3, 9, 56, 51\}$. L'albero in Figura 1 è un albero ternario di ricerca che rappresenta K .

Sia x il nodo corrispondente al 56 più vicino alla radice. Allora nel nodo x , *left* punta a un albero radicato in 78, *right* punta a un albero radicato in 51, *mid* punta a un albero radicato in 56. Quest'ultimo nodo ha come *mid* un nuovo albero radicato in 56, NULL come *left* e NULL come *right*.

Assegnamento

Implementare le seguenti funzioni (i cui prototipi sono anch'essi contenuti nel file *ttree.h*).

¹Si noti che, dato che K può contenere interi ripetuti, più nodi dell'albero possono contenere lo stesso valore

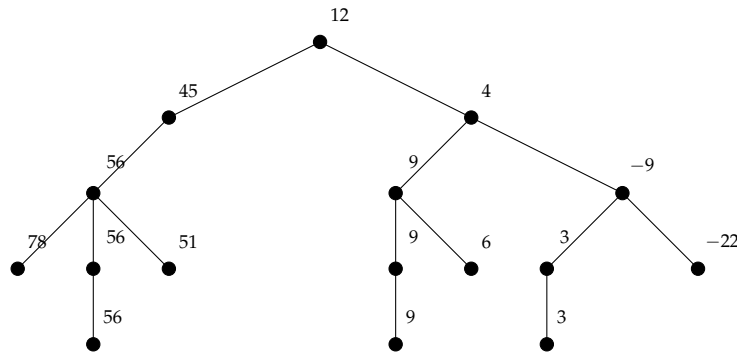


Figure 1: Albero Ternario di ricerca per il multi-insieme $\{12, 45, 56, 78, 4, 56, 9, 6, 9, -9, 3, -22, 3, 9, 56, 51\}$

1. `void inserisci (int x, nodo_t ** proot);`

Inserisce una nuova chiave x nell'albero mantenendo l'ordinamento nell'albero radicato in *proot* (puntatore al puntatore alla radice del albero). La procedura di inserimento controlla se la chiave x è minore, uguale o maggiore rispetto alla chiave di *proot* e richiama ricorsivamente se stessa rispettivamente su *L*, *M* oppure *R*.

L'albero in Figura 1 è stato costruito inserendo le chiavi nell'ordine dato usando la funzione `inserisci` dettagliata nella prossima sezione. Ad esempio, quando $K = \{12, 45, 56, 78, 4, 56, 9, 6, 9, -9, 3, -22, 3, 9\}$ e si aggiunge il 56, questo viene confrontato con 12, con 45, con 56, con 56 e poi inserito in quest'ultimo. Quando si inserisce il 51, questo viene confrontato con il 12, il 45, il 56 e poi inserito in quest'ultimo.

2. `void cancella (int x, nodo_t ** proot);`

Cancella una sola occorrenza della chiave x (se presente) nell'albero radicato in *proot* mantenendo l'ordinamento. Dopo la cancellazione, l'albero deve continuare a soddisfare le proprietà di un albero di ricerca ternario. Per cui:

- Se la chiave corrisponde a una foglia questa viene cancellata direttamente.
- Se il sottoalbero di destra (risp. sinistra) è vuoto e quello di sinistra (risp. destra) non è vuoto, il nodo viene sostituito con il sottoalbero di sinistra (risp. destra).
- Altrimenti troviamo il valore maggiore contenuto nel sottoalbero di destra e lo sostituiamo al nodo.

3. `void free_albero (nodo_t ** root);`

Libera tutta la memoria occupata dall'albero puntato da *proot* (puntatore a puntatore alla radice). La radice va settata a NULL all'interno della funzione.

4. `int scrivi_albero(FILE* f, nodo_t* root);`

Esporta l'albero puntato da *root* su file *f* in un formato a caratteri a scelta dello studente. **Il formato deve essere definito e motivato in modo non ambiguo in una relazione da consegnare insieme al codice.** Quando la funzione viene invocata, si assuma che il file *f* passato sia già aperto in scrittura. La funzione ritorna 0 se tutto è andato a buon fine, -1 altrimenti.

5. `int leggi_albero(FILE* f, nodo_t** root);`

Questa funzione legge l'albero dal file *f*, espresso nel formato definito nella relazione menzionata al punto precedente, e costruisce l'albero. *root* punta alla radice dell'albero così costruito. La funzione ritorna 0 se tutto è andato a buon fine, -1 altrimenti.

6. `void ordine (nodo_t * proot);`

Stampare tutti i numeri interi distinti x contenuti in K in ordine crescente (se x occorre più volte in K deve essere stampato una sola volta). Nell'albero in Figura 1, bisogna stampare:

-22
-9
3
4
6
9
12
45
51
56
78

7. `void cerca1 (nodo_t * root);`

Dato un nodo y , sia $M(y)$ il numero di nodi nel sottoalbero M di y e $L(y)$ il numero di nodi nel sottoalbero L di y . Stampare le chiavi di tutti i nodi y tali che $M(y) < L(y)$. Nell'albero in Figura 1, bisogna stampare (in qualsiasi ordine):

-9
4
45
12

8. `void cerca2 (nodo_t * root);`

Dato un nodo y , sia $A(y)$ la media dei valori contenuti nel sottoalbero radicato in y (compreso il valore in y). Stampare le chiavi di tutti i nodi y tali che $A(y) < 5$. Nell'albero in Figura 1, bisogna stampare (in qualsiasi ordine):

-22
3
3
-9
4

9. `void cerca3 (nodo_t * proot);`

Dato un nodo y , sia $H(y)$ il suo livello, $L(y)$ il numero di nodi contenuti nel suo sottoalbero L , $R(y)$ il numero di nodi contenuti nel suo sottoalbero R .² Stampare le chiavi dei nodi y tali che $L(y) + R(y) + H(y)$ è multiplo di 3. Nell'albero in Figura 1, bisogna stampare (in qualsiasi ordine):

-22
3
6
9
9
4
51

²La radice ha livello 0, i suoi figli hanno livello 1, e così via.

56
78
45
12

Le funzioni ai punti 6, 7, 8, 9 devono avere complessità lineare e stampare una chiave per riga.